QT

Team Emertxe

Fundamentals of Qt Module

Fundamentals of Qt

- The Story of Qt
- Developing a Hello World Application
- Hello World using Qt Creator
- Practical Tips for Developers

Objectives

- About the history of Qt
- About Qt's ecosystem
- A high-level overview of Qt
- How to create first hello world program
- Build and run a program cross platform
- To use Qt Creator IDE
- Some practical tips for developing with Qt

Fundamentals of Qt

- The Story of Qt
- Developing a Hello World Application
- Hello World using Qt Creator
- Practical Tips for Developers

Some history

- **Qt Development Frameworks founded in 1994**
  - Trolltech acquired by Nokia in 2008
  - Qt Commercial business acquired by Digia in 2011
  - Qt business acquired by “Digia Qt” from Nokia in 2012
  - “Digia, Qt” becomes its own entity “The Qt Company”
  - Trusted by over 6,500 companies worldwide
- **Qt: a cross-platform application and UI framework**
  - For desktop, mobile and embedded development
  - Used by more than 350,000 commercial and open source developers
  - Backed by Qt consulting, support and training

Qt is everywhere

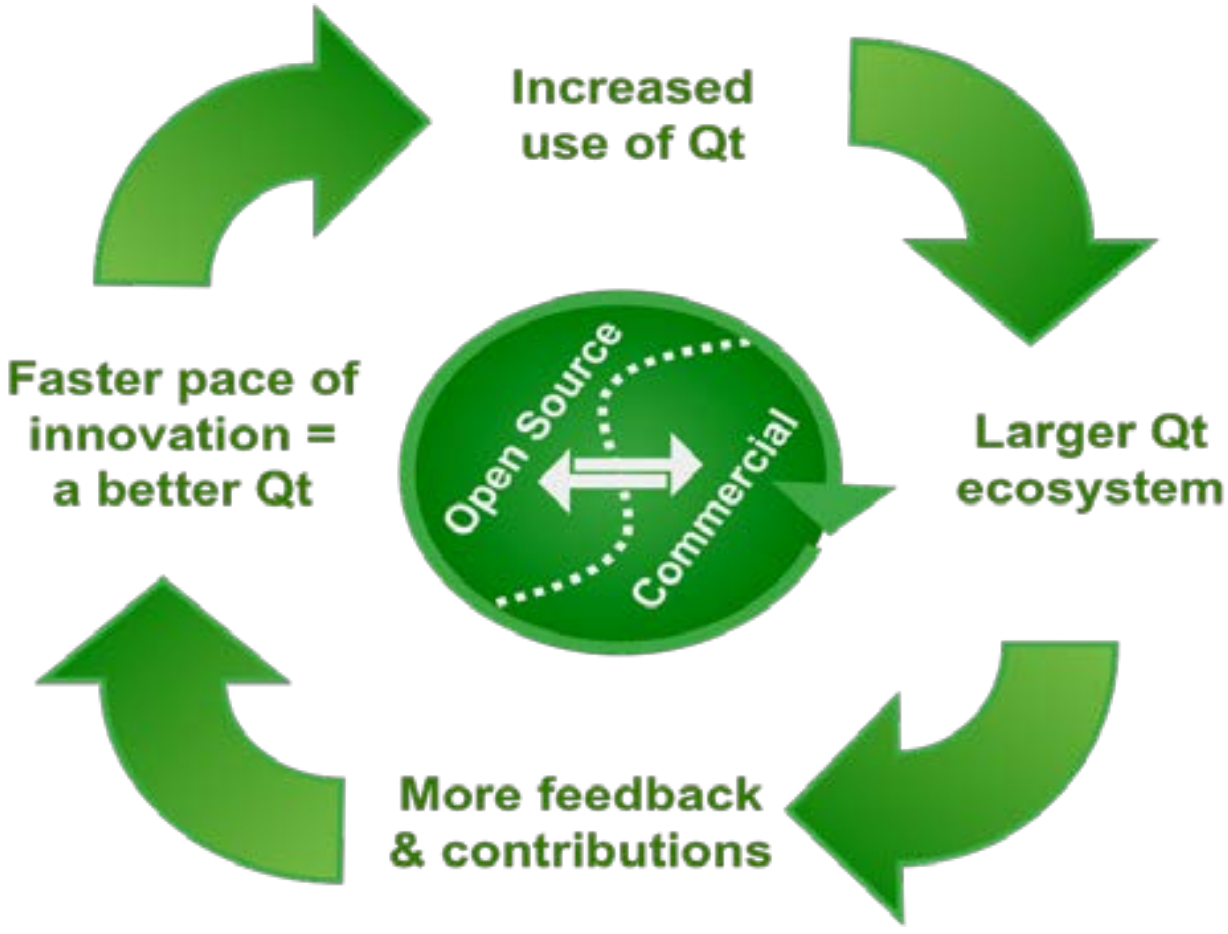
From embedded devices to desktop applications

By companies from many industries

AMD, Google, Canon, Alcatel-Lucent, BT, DAIMLER, EPSON, esa, freescale semiconductor, HITACHI Inspire the Next, GENERAL DYNAMICS C4 Systems, Mentor Graphics, PGS, HONDA The Power of Dreams, MITSUBISHI, NORTEL NETWORKS, ORACLE, PHILIPS, Pioneer, skype, Raytheon, Schlumberger, ROHDE&SCHWARZ, SCANIA, SENNHEISER, SHARP, xerox, VOLVO, vodafone, THOMSON REUTERS

ΣMERTXE

Qt virtuous cycle



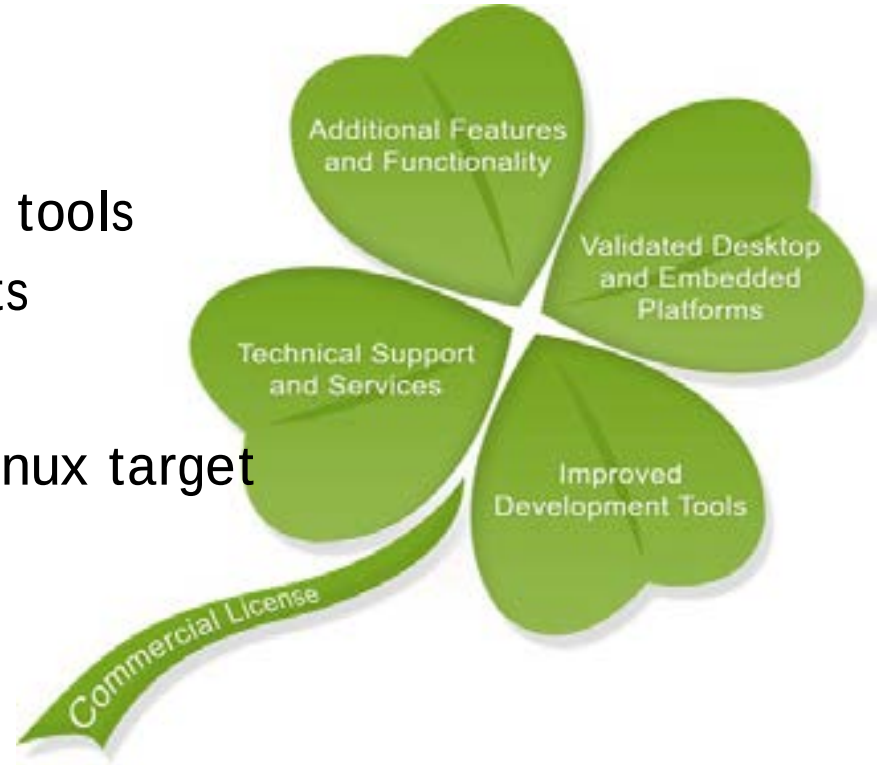
EMERTXE

Why Qt

- Write code once to target multiple platforms
- Produce compact, high-performance applications
- Focus on innovation, not infrastructure coding
- Choose the license that fits you
  - Commercial, LGPL or GPL
- Count on professional services, support and training
- Take part in an active Qt ecosystem

Qt Commercial

- Improved development tools for increased productivity and tangible cost savings
- Qt Commercial SDK
  - All Qt Commercial libraries and tools
  - Additional tools and components
- Qt Creator Embedded Target
  - Deploy directly to embedded Linux target
- RTOS toolchain integration
- Visual Studio Add-On



Qt5

- Awesome graphics capabilities
  - OpenGL as a standard feature of user interfaces
  - Shader-based graphics effects in QtQuick 2
- New modular structure
  - Qt Essential modules available on all platforms
  - Add-on modules provide additional or platform-specific functionality
- Developer productivity and flexibility
  - More web-like development with QtQuick 2, V8 JavaScript engine, and Qt JSON DB
- Cross-platform portability
  - Qt Platform Abstraction (QPA) replaces QWS and platform ports

Qt5 Modules

EMERTXE

Qt Demo

- Let's have a look at the QtDemo Application
- Comes with every Qt installation

Technology	Demo
Painting	*Demonstrations/Path Stroking*
Widgets	*Demonstrations/Books*
Widgets	*Demonstrations/TextEdit*
GraphicsView	*Demonstrations/40.000 Chips*
OpenGL	*Demonstrations/Boxes*
WebKit	*Demonstrations/Browser*

Fundamentals of Qt

- The Story of Qt
- Developing a Hello World Application
- Hello World using Qt Creator
- Practical Tips for Developers

“Hello world” in Qt

```
// main.cpp
#include <QtWidgets>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton  button("Hello world");
    button.show();
    return app.exec();
}
```

- Program consists of
  - main.cpp- application code
  - helloworld.pro- project file
- Hello world demo

Project File

helloworld.pro

- helloworld.pro file
  - lists source and header files
  - provides project configuration

```
# File: helloworld.pro
SOURCES  = main.cpp

HEADERS +=   # No headers used
QT = core gui widgets
```

- Assignment to variables
  - Possible operators =, +=, -=
- Qmake-tutorial

EMERTXE

Using qmake

- qmake tool
  - Creates cross-platform makefiles
- Build project using qmake

```
cd helloworld
qmake helloworld.pro    # creates Makefile
Make                    # compiles and links application
./helloworld            # executes application
```

- Tip: qmake -project
  - Creates default project file based on directory content
- Qmake manual

Qt Creator does it all for you

Fundamentals of Qt

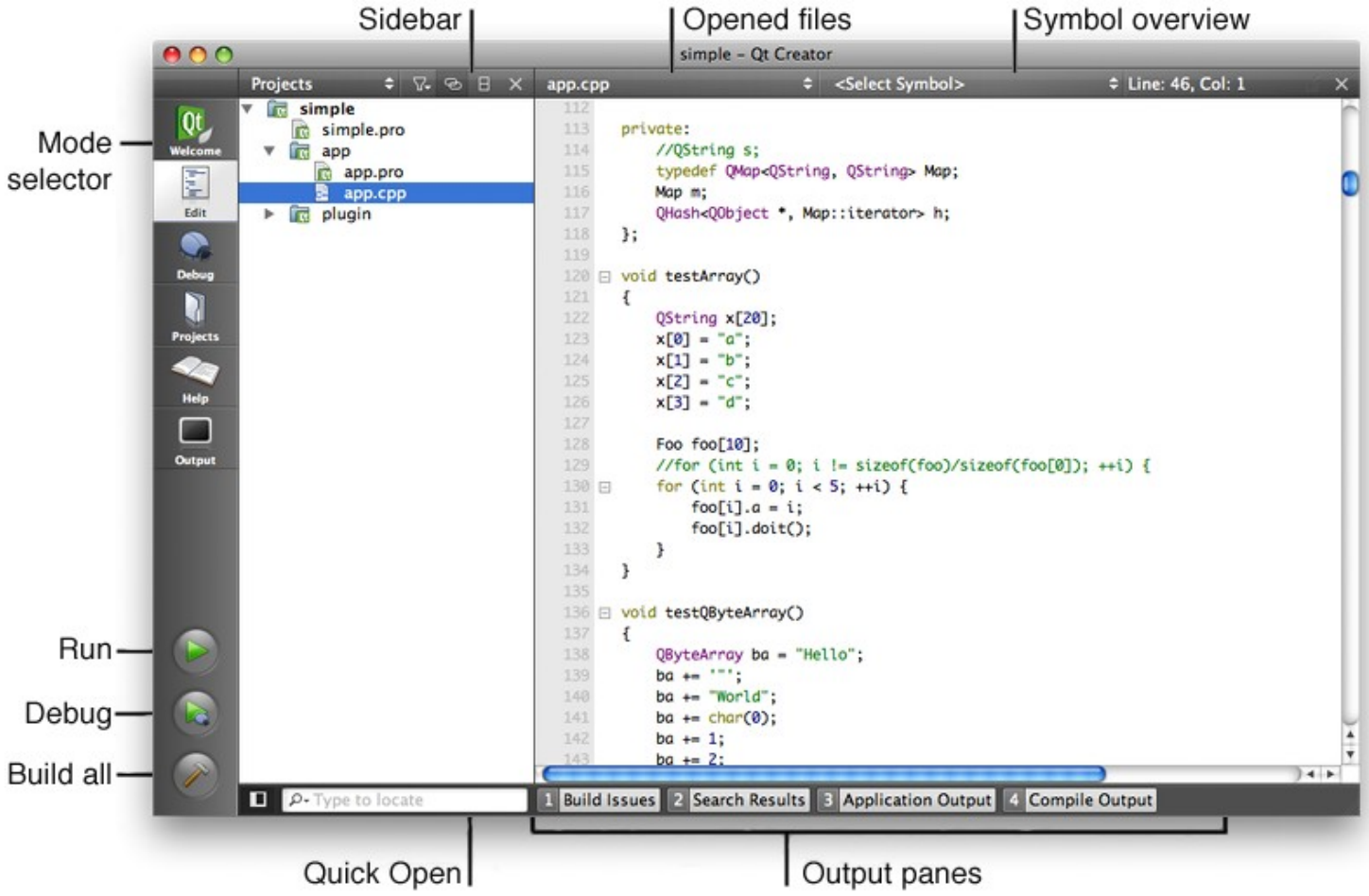
- The Story of Qt
- Developing a Hello World Application
- Hello World using Qt Creator
- Practical Tips for Developers

QtCreator IDE

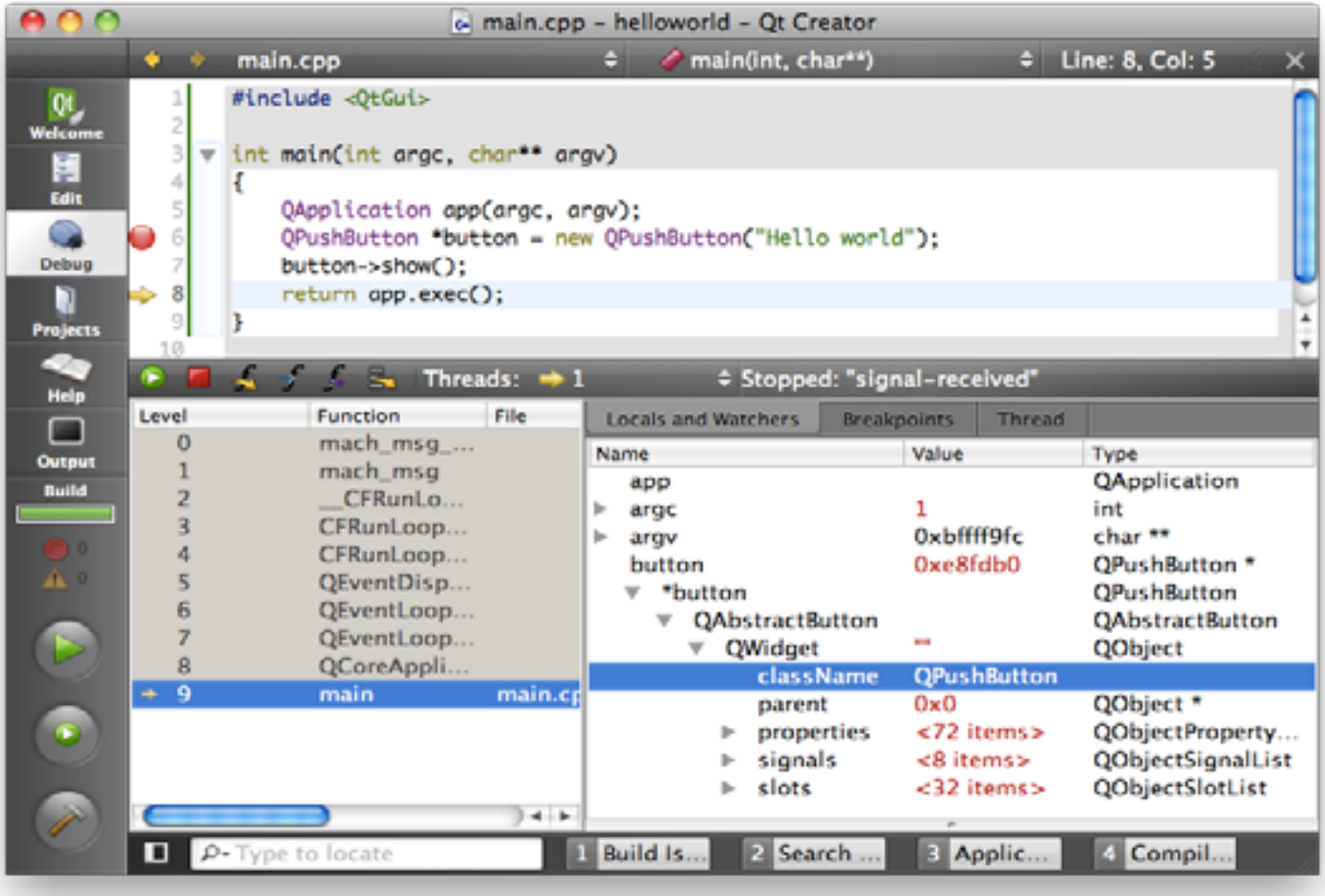
- Advanced C++ code editor
- Integrated GUI layout and forms designer
- Project and build management tools
- Integrated, context-sensitive help system
- Visual debugger
- Rapid code navigation tools
- Supports multiple platforms

Qt Creator

Components



Debugging

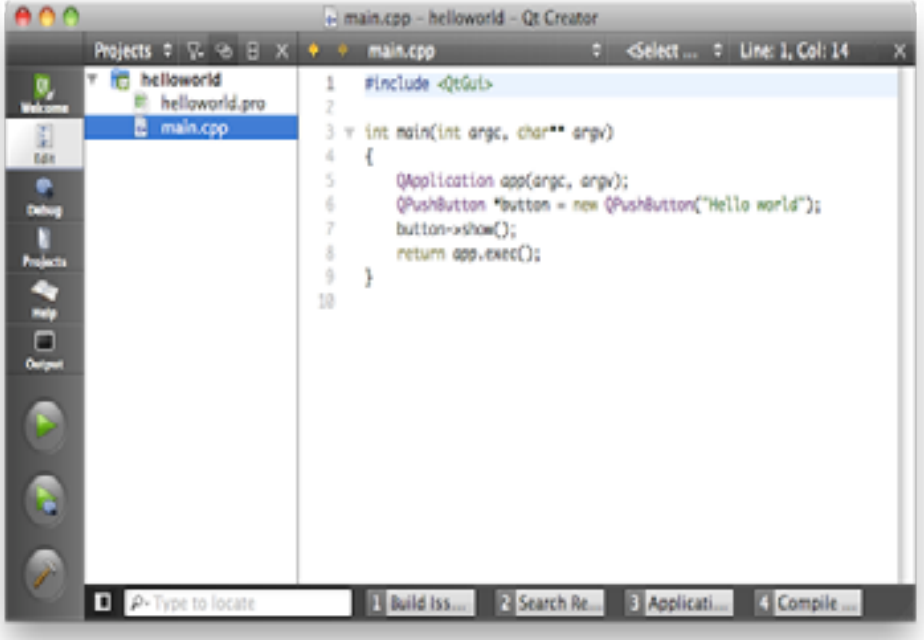


EMERTXE

Qt Creator Demo

"Hello World"

- Creation of an empty Qt Widget project
- Adding the main.cpp source file
- Writing of the Qt Hello World Code
- Running the application
- Debugging the application



EMERTXE

Fundamentals of Qt

- The Story of Qt
- Developing a Hello World Application
- Hello World using Qt Creator
- Practical Tips for Developers

C++ knowledge

- Objects and classes
  - Declaring a class, inheritance, calling member functions etc.
- Polymorphism
  - That is virtual methods
- Operator overloading

Qt Documentation

- Reference Documentation
  - All classes documented
  - Contains tons of examples
- Collection of Howto's and Overviews
- A set of Tutorials for Learners

ΣMERTXE

Modules and

Include files

- Qt Modules
  - QtCore, QtGui, QtWidgets, QtXml, QtSql, QtNetwork, QtTest .. See documentation
- Enable Qt Module in qmake .profile:
  - `QT += network`
- Default: qmake projects use QtCore and QtGui
  - Any Qt class has a header file.

```
#include <QLabel>
#include <QtWidgets/QLabel>
```

  - Any Qt Module has a header file.

```
#include <QtGui>
```

Includes and
Compilation Time

Module includes

```
#include <QtGui>
```

- Precompiled header(PCH) and the compiler
  - If **not** supported may add extra compile time
  - If supported may speed up compilation
  - Supported on: Windows, Mac OS X, Unix

Class includes

```
#include <QLabel>
```

- Reduce compilation time
  - Use class includes (`#include <QLabel>`)
  - Forward declarations (classQLabel;)

Place module includes before other includes.

Summary

- What is Qt
- Which code lines do you need for a minimal Qt application?
- What is a .pro file?
- What is qmake, and when is it a good idea to use it?
- What is a Qt module and how to enable it in your project?
- How can you include a QLabel from the QtGui module?
- Name places where you can find answers about Qt problems

Core classes

Core classes

- Qt's Object Model
  - QObject
  - Qwidget
- String Handling
- Container Classes

Objectives

- What is Qt object model?
- The basics of the widget system in C++
- Which utility classes exists to help you in C++

Core classes

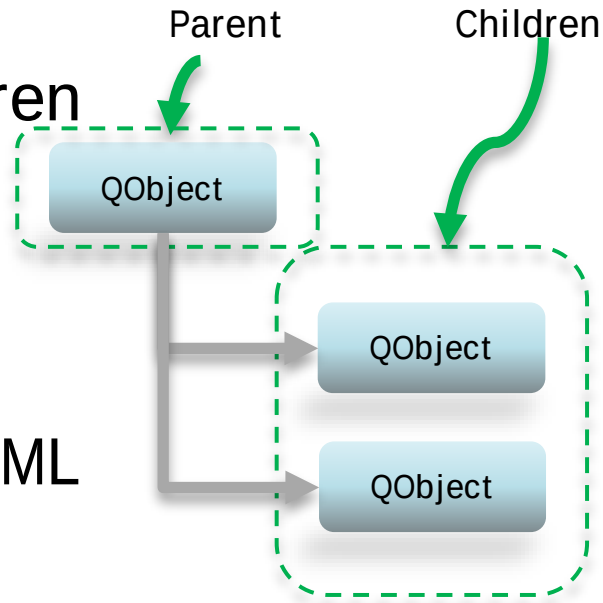
- Qt's Object Model
  - QObject
  - QWidget
- String Handling
- Container Classes

QObject

- QObject is the heart of Qt's object model
- Include these features:
  - Memory management
  - Object properties
  - Signals and slots
  - Event handling
- QObject has no visual representation

Object tree

- QObjects organize themselves in object trees
  - Based on parent-child relationship
- QObject(QObject *parent = 0)
  - Parent adds object to list of children
  - Parent owns children
- Widget Centric
  - Used intensively with QtWidget
  - Less so when using Qt/C++ from QML



Note: Parent-child relationship is NOT inheritance

Creating Objects

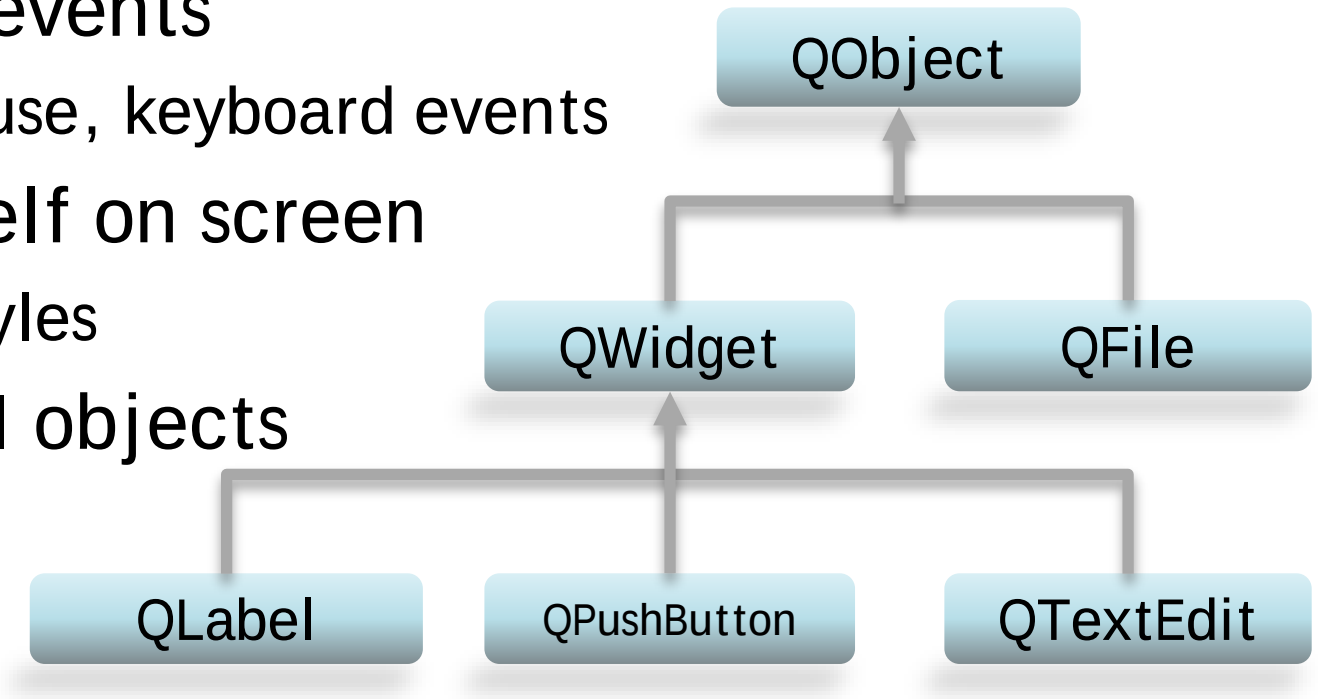
- **On Heap** - QObject with parent:

```
QTimer* timer = new QTimer(this);
```

  - Parent takes ownership.
  - Copy is disabled
- **On Stack** - value types
  - QString, QStringList, Qcolor
- **On Stack** - QObject without parent:Exceptions
  - QFile, Qapplication (Usually allocated on the stack)
  - Top level QWidgets: QMainWindow
- **Stack or Heap** - QDialog - depending on lifetime

QWidget

- Derived from QObject
  - Adds visual representation
- Receives events
  - e.g. mouse, keyboard events
- Paints itself on screen
  - Using styles
- Base of UI objects



QWidget

- new QWidget(0)
  - Widget with no parent="window"
- QWidget's children
  - Positioned in parent's coordinate system
  - Clipped by parent's boundaries
- QWidget parent
  - Propagates state changes
    - hides/shows children when it is hidden/shown itself
    - enables/disables children when it is enabled/disabled itself

Widgets contain other widgets

- Container Widget
  - Aggregates other child-widgets
- Use layouts for aggregation
  - In this example: `QHBoxLayout` and `QVBoxLayout`
  - Note: Layouts are *not* widgets
- Layout Process
  - Add widgets to layout
  - Layouts may be nested
  - Set layout on container widget

Container Widget

```
// container (window) widget creation
QWidget* container = new QWidget;
QLabel* label = new QLabel("Note:", container);
QTextEdit* edit = new QTextEdit(container);
QPushButton* clear = new QPushButton("Clear", container);
QPushButton* save = new QPushButton("Save", container);
// widget layout
QVBoxLayout* outer = new QVBoxLayout();
outer->addWidget(label);
outer->addWidget(edit);
QHBoxLayout* inner = new QHBoxLayout();
inner->addWidget(clear);
inner->addWidget(save);
container->setLayout(outer);
outer->addLayout(inner); // nesting layouts
```

Demo

ΣMERTXE

Core classes

- Qt's Object Model
  - QObject
  - QWidget
- String Handling
- Container Classes

QString

Strings can be created in a number of ways:

- Conversion constructor and assignment operators:

```
QString str("abc");
str = "def";
```

- From a number using a static function:

```
QString n = QString::number(1234);
```

- From a char pointer using the static functions:

```
QString text = QString::fromLatin1("Hello Qt");
QString text = QString::fromUtf8(inputText);
QString text = QString::fromLocal8Bit(cmdLineInput);
QString text = QStringLiteral("Literal String");
(Assumed to be UTF-8)
```

- From char pointer with translations:

```
QString text = tr("Hello Qt");
```

QString

- operator+ and operator+=

```
QString str = str1 + str2;
fileName += ".txt";
```

- simplified() // removes duplicate whitespace
- left(), mid(), right() // part of a string
- leftJustified(), rightJustified() // padded version

```
QString s = "apple";
QString t = s.leftJustified(8, '.');
// t == "apple..."
```

QString

Data can be extracted from strings.

- Numbers:

```
QString text = ...;
int value = text.toInt();
float value = text.toFloat();
```

- Strings:

```
QString text = ...;
QByteArray bytes = text.toLatin1();
QByteArray bytes = text.toUtf8();
QByteArray bytes = text.toLocal8Bit();
```

QString::arg()

```
int i = ...;
int total = ...;
QString fileName = ...;
QString status = tr("Processing file %1 of %2: %3")
.arg(i).arg(total).arg(fileName);
double d = 12.34;
QString str = QString::fromLatin1("delta: %1").arg(d,0,'E',3);
// str == "delta: 1.234E+01"
```

- Safer: arg(QString,...,QString) (“multi-arg()”).
- But: only works with QString arguments.

QString

- Obtaining raw character data from a QByteArray:

```
char *str = bytes.data();
const char *str = bytes.constData();
```

WARNING:

- Character data is only valid for the lifetime of the byte array.
- Calling a non-const member of bytes also invalidates ptr.
- Either copy the character data or keep a copy of the byte array.

QString

- `length()`
- `endsWith()` and `startsWith()`
- `contains()`, `count()`
- `indexOf()` and `lastIndexOf()`

Expression can be characters, strings, or regular expressions

QString

- `QString::split(), QStringList::join()`
- `QStringList::replaceInStrings()`
- `QStringList::filter()`

QString

- QRegExp supports
  - Regular expression matching
  - Wildcard matching
- QString cap(int)

  QStringList capturedTexts()

```
QRegExp rx("^\\d\\d?$"); // match integers 0 to 99
rx.indexIn("123"); // returns -1 (no match)
rx.indexIn("-6"); // returns -1 (no match)
rx.indexIn("6"); // returns 0 (matched as position 0)
```

- Regular expression documentation

Core classes

- Qt's Object Model
  - QObject
  - QWidget
  - Variants
  - Properties
- String Handling
- Container Classes

Container Classes

General purpose template-based container classes

- **QList<QString>** - *Sequence Container*
  - Other: QLinkedList, QStack , QQueue ...
- **QMap<int**, **QString>** - *Associative Container*
  - Other: QHash, QSet, QMultiMap, QMultiHash

Qt's Container Classes compared to STL(standard template library)

- Lighter, safer, and easier to use than STL containers
- If you prefer STL, feel free to continue using it.
- Methods exist that convert between Qt and STL
  - e.g. you need to pass std::list to a Qt method

Container Classes

Using QList

```
QList<QString> list;
list << “one” << “two” << “three”;
QString item1 = list[1]; // “two”
for(int i=0; i<list.count(); i++) {
const QString &item2 = list.at(i);
}
int index = list.indexOf("two"); // returns 1
```

Using QMap

```
QMap<QString, int> map;
map[“Norway”] = 5; map[“Italy”] = 48;
int value = map[“France”]; // inserts key if not exists
if(map.contains(“Norway”)) {
int value2 = map.value(“Norway”); // recommended lookup
}
```

Complexity

How fast is a function when number of items grow

Sequential Container

	Lookup	Insert	Append	Prepend
QList	O(1)	O(n)	O(1)	O(1)
QVector	O(1)	O(n)	O(1)	O(n)
QLinkedList	O(n)	O(1)	O(1)	O(1)

Complexity

Associative Container

	Lookup	Insert
Qmap	O(log(n))	O(log(n))
QHash	O(1)	O(1)

all complexities are amortized

Classes
in Container

- Class must be an *assignable data type*
- Class is *assignable*, if:

```
class Contact {
public:
Contact() {} // default constructor
Contact(const Contact &other); // copy
constructor
// assignment operator
Contact &operator=(const Contact &other);
};
```

- *If copy constructor or assignment operator is not provided*
  - C++ will provide one (uses member copying)
- *If no constructors provided*
  - Empty default constructor provided by C++

Container Keys

- Type K as key for QMap:
  - bool K::operator<(const K&) or

  bool operator<(const K&, const K&)

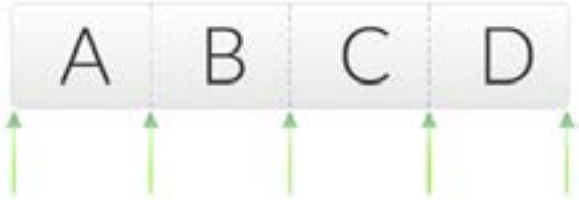
```
bool Contact::operator<(const Contact& c);
bool operator<(const Contact& c1, const Contact& c2);
```

  - QMap Documentation
- Type K as key for QHash or QSet:
  - bool K::operator==(const K&) or bool operator==(const K&, const K&)
  - uint qHash(const K&)
  - QHash Documentation

Iterators

- Allow reading a container's content sequentially
- **Java-style iterators**: simple and easy to use
  - `QListIterator<...>`for read
  - `QMutableListIterator<...>`for read-write
- **STL-style iterators** slightly more efficient
  - `QList::const_iterator` for read
  - `QList::iterator()` for read-write
- Same works for QSet, QMap, QHash, …

Iterators
Java style



- Example QList iterator
```
QList<QString> list;
list << "A" << "B" << "C" << "D“;
QListIterator<QString> it(list);
```
- Forward iteration
```
while(it.hasNext()) {
qDebug() << it.next(); // A B C D
}
```
- Backward iteration
```
it.toBack(); // position after the last item
while(it.hasPrevious()) {
qDebug() << it.previous(); // D C B A
}
```
- QListIterator Documentation

Modifying During Iteration

- Use *mutable* versions of the iterators
  - e.g. QMutableListIterator.

```
QList<int> list;
list << 1 << 2 << 3 << 4;
QMutableListIterator<int> i(list);
while (i.hasNext()) {
if (i.next() % 2 != 0)
i.remove();
}
// list now 2, 4
```

- remove() and setValue()
  - Operate on items just jumped over using next()/previous()
- insert()
  - Inserts item at current position in sequence
  - previous() reveals just inserted item

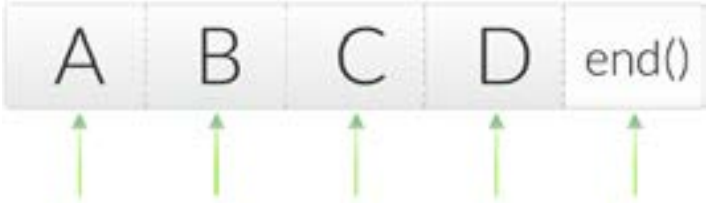
QMap and QHash

- next() and previous()
  - Return Item class with key() and value()
- Alternatively use key() and value() from iterator

```
QMap<QString, QString> map;
map[“Paris”] = “France”;
map[“Guatemala City”] = “Guatemala”;
map[“Mexico City”] = “Mexico”;
map[“Moscow”] = “Russia”;
QMutableMapIterator<QString, QString> i(map);
while (i.hasNext()) {
if (i.next().key().endsWith(“City”))
i.remove();
}
// map now "Paris", "Moscow“
```

- Demo

Iterators
STL-style



- Example QList iterator
```
QList<QString> list;
list << "A" << "B" << "C" << "D";
QList<QString>::iterator i;
```
- Forward mutable iteration
```
for (i = list.begin(); i != list.end(); ++i) {
*i = (*i).toLower();
}
```
- Backward mutable iteration
```
i = list.end();
while (i != list.begin()) {
--i;
*i = (*i).toLower();
}
```
- **QList<QString>**::const_iterator for read-only

foreach

- It is a macro, feels like a keyword

  foreach (*variable*, *container*) *statement*

```
foreach (const QString& str, list) {
if (str.isEmpty())
break;
qDebug() << str;
}
```

- Break and continue as normal
- Modifying the container while iterating
  - results in container being copied
  - iteration continues in unmodified version
- Not possible to modify item
  - iterator variable is a const reference.

Qt algorithms

- **qSort**(begin, end) sort items in range
- **qFind**(begin, end, value) find value
- **qEqual**(begin1, end1, begin2) checks two ranges
- **qCopy**(begin1, end1, begin2) from one range to another
- **qCount**(begin, end, value, n) occurrences of value in range
- For parallel (ie. multi-threaded) algorithms
  - QtConcurrent()
- QtAlgorithms Documentation

Examples

- Counting 1's in list

```
QList<int> list;
list << 1 << 2 << 3 << 1;
int count = 0;
qCount(list, 1, count); // count the 1's
qDebug() << count; // 2 (means 2 times 1)
```

- Copy list to vector

```
QList<QString> list;
list << "one" << "two" << "three";
QVector<QString> vector(3);
qCopy(list.begin(), list.end(),
vector.begin());
// vector: [ "one", "two", "three" ]
```

Examples

- Case insensitive sort

```
bool lessThan(const QString& s1, const QString&
s2) {
return s1.toLower() < s2.toLower();
}
// ...
QList<QString> list;
list << "AlPha" << "beTA" << "gamma" << "DELTA";
qSort(list.begin(), list.end(), lessThan);
// list: [ "AlPha", "beTA", "DELTA", "gamma" ]
```

Implicit Sharing

If an object is copied, then its data is copied *only when* the data of one of the objects is changed

- Shared class has a pointer to shared data block
  - Shared data block = reference counter and actual data
- Assignment is a shallow copy
- Changing results into deep copy (detach)

```
QList<int> l1, l2; l1 << 1 << 2;
l2 = l1; // shallow-copy: l2 shares date with l1
l2 << 3; // deep-copy: change triggers detach from l1
```

Important to remember when inserting items into a container, or when returning a container.

Summary

- Explain how object ownership works in Qt?
- What are the key responsibilities of QObject?
- What does it mean when a QWidget has no parent?
- What is the purpose of the class QVariant?
- What do you need to do to support properties in a class?
- Name the different ways to go through the elements in a list, and
- Discuss advantages and disadvantages of each method.

Widgets

Widgets

- Common Widgets
- Layout Management
- Custom Widgets

Widgets

- **Common Widgets**
  - Text widgets
  - Value based widgets
  - Organizer widgets
  - Item based widgtes
- **Layout Management**
  - Geometry management
  - Advantages of layout managers
  - Qt's layout managers
  - Size policies
- **Custom Widgets**
  - Rules for creating own widgets

Widgets

- Common Widgets
- Layout Management
- Custom Widgets

Text Widgets

- **QLabel**

```
label = new QLabel("Text", parent);
```

  - `setPixmap( pixmap )` - as content

- **QLineEdit**

```
line = new QLineEdit(parent);
line->setText("Edit me");
line->setEchoMode(QLineEdit::Password);
connect(line, SIGNAL(textChanged(QString)) ...
connect(line, SIGNAL(editingFinished()) ...
```

- **QTextEdit**

```
edit = new QTextEdit(parent);
edit->setPlainText("Plain Text");
edit->append("<h1>Html Text</h1>");
connect(edit,SIGNAL(textChanged(QString)) …
```

ΣMERTXE

Button widgets

- **QAbstractButton**

  Abstract base class of buttons
- **QPushButton**

```
button = new QPushButton("Push Me", parent);
button->setIcon(QIcon("images/icon.png"));
connect(button, SIGNAL(clicked()) ...
setCheckable(bool) - toggle button
```
- **QRadioButton**

```
radio = new QRadioButton("Option 1", parent);
```
- **QCheckBox**

```
check = new QCheckBox("Choice 1", parent);
```

Value Widgets

- **QSlider**

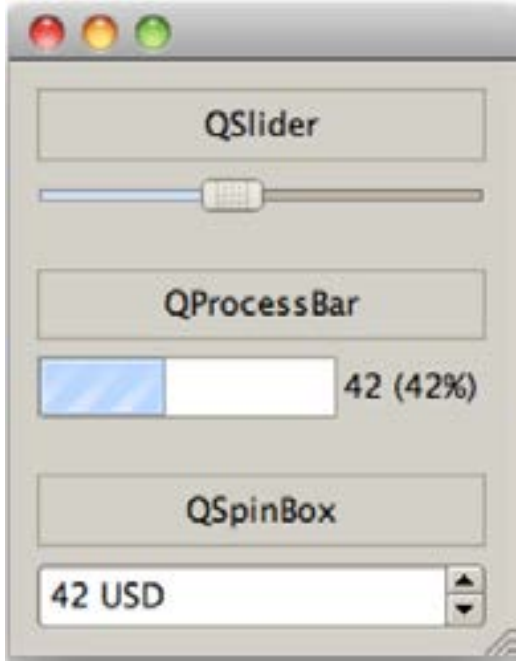
```
slider = new QSlider(Qt::Horizontal, parent);
slider->setRange(0, 99);
slider->setValue(42);
connect(slider, SIGNAL(valueChanged(int)) ...
```

- **QProgressBar**

```
progress = new QProgressBar(parent);
progress->setRange(0, 99);
progress->setValue(42);
// format: %v for value; %p for percentage
progress->setFormat("%v (%p%)");
```

- **QSpinBox**

```
spin = new QSpinBox(parent);
spin->setRange(0, 99);
spin->setValue(42);
spin->setSuffix(" USD");
connect(spin,SIGNAL(valueChanged(int)) ...
```



ΣMERTXE

Organizer Widgets

- **QGroupBox**

```
box = new QGroupBox("Your Options", parent);
// ... set layout and add widgets
```

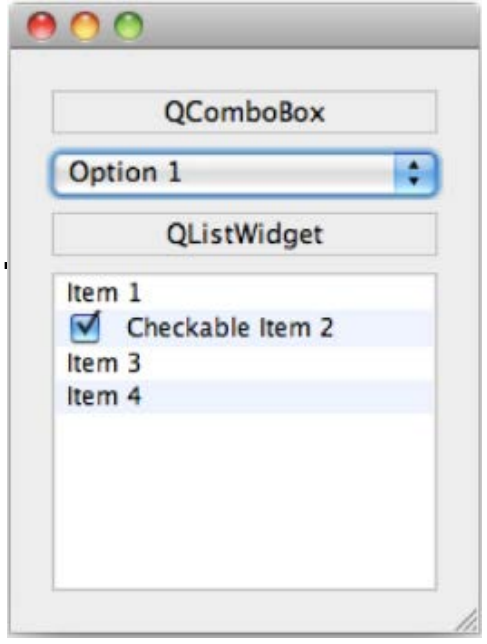
setCheckable(bool) - checkbox in title

- **QTabWidget**

```
tab = new QTabWidget(parent);
tab->addWidget(widget, icon, "Tab 1");
connect(tab, SIGNAL(currentChanged(int)) ...
```

  - setCurrentWidget(widget)
    - Displays page assoziated by widget
  - setTabPosition(position)
    - Defines where tabs are drawn
  - setTabsClosable(bool)
    - Adds close buttons

Item Widgets



- **QComboBox**

```
combo = new QComboBox(parent);
combo->addItem("Option 1", data);
connect(combo, SIGNAL(activated(int)) ...
QVariant data = combo->itemData(index);
```

  - setCurrentIndex(index)
- **QListWidget**

```
list = new QListWidget(parent);
list->addItem("Item 1");
// ownership of items with list
item = new QListWidgetItem("Item 2", list);
item->setCheckState(Qt::Checked);
connect(list, SIGNAL(itemActivated(QListWidgetItem*)) .
```

- Other Item Widgets: QTableWidget, QTreeWidget

Other Widgets

- **QToolBox**
  Column of tabbed widget items
- **QDateEdit, QTimeEdit, QDateTimeEdit**
  Widget for editing date and times
- **QCalendarWidget**
  Monthly calendar widget
- **QToolButton**
  Quick-access button to commands
- **QSplitter**
  Implements a splitter widget
- **QStackedWidget**
  Stack of widgets
  Only one widget visible at a time
- Widget Documentation

Widgets

- Common Widgets
- Layout Management
- Custom Widgets

Layout widgets

Definition

Layout: Specifying the relations of elements to each other instead of the absolute positions and sizes.

- Advantages:
  - Works with different languages.
  - Works with different dialog sizes.
  - Works with different font sizes.
  - Better to maintain.
- Disadvantage
  - Need to design your layout first.

Layout widgets

- Place and resize widgets (Without layouts)
  - move()
  - resize()
  - setGeometry()
- Example:

```
QWidget *parent = new QWidget(...);
parent->resize(400,400);
QCheckBox *cb = new QCheckBox(parent);
cb->move(10, 10);
```

Layout widgets

- On layout managed widgets never call setGeometry(), resize(), or move()
- Preferred
  - Override
  - sizeHint()
  - minimumSizeHint()
- Or call
  - setFixedSize()
  - setMinimumSize()
  - setMaximumSize()

Layout classes

- **QHBoxLayout**
  - Lines up widgets horizontally
- **QVBoxLayout**
  - Lines up widgets vertically
- **QGridLayout**
  - Arranges the widgets in a grid
- **QFormLayout**
  - Lines up a (label, widget) pairs in two columns.
- **QStackedLayout**
  - Arranges widgets in a stack
    - only topmost is visible

QHBoxLayout & QVBoxLayout

- Lines up widgets horizontally or vertically
- Divides space into boxes
- Each managed widgets fills in one box


```
QWidget* window = new QWidget;
QPushButton* one = new QPushButton("One");
...
QHBoxLayout* layout = new QHBoxLayout;
layout->addWidget(one);
...
window->setLayout(layout);
```

QGridLayout

```
QWidget* window = new QWidget;
QPushButton* one = new QPushButton("One");
QGridLayout* layout = new QGridLayout;
layout->addWidget(one, 0, 0); // row:0, col:0
layout->addWidget(two, 0, 1); // row:0, col:1
// row:1, col:0, rowSpan:1, colSpan:2
layout->addWidget(three, 1, 0, 1, 2);
window->setLayout(layout)
```

Demo

- Additional
  - `setColumnMinimumWidth()` (minimum width of column)
  - `setRowMinimumHeight()` (minimum height of row)

No need to specify rows and columns before adding children.

QFormLayout

- A two-column layout
  - Column 1 a label (as annotation)
  - Column 2 a widget (as field)
- Respects style guide of individual platforms.

```
QWidget* window = new QWidget();
QPushButton* one = new QPushButton("One");
...
QFormLayout* layout = new QFormLayout();
layout->addRow("One", one);
...
window->setLayout(layout)
```

- Form layout with clean looks and mac style

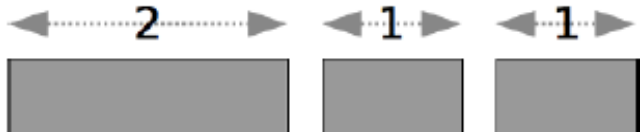
Hands-on

Lab 3 Contact form

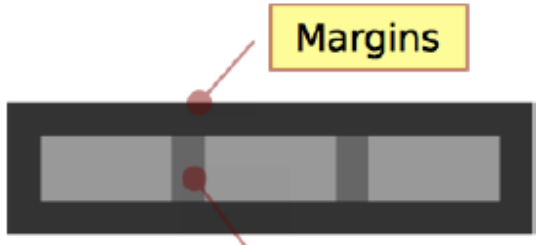
- Objectives
- Template code

EMERTXE

Layout Terms

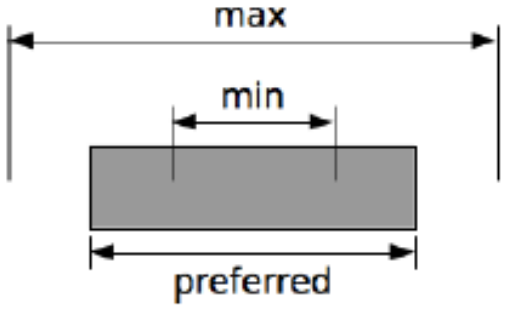


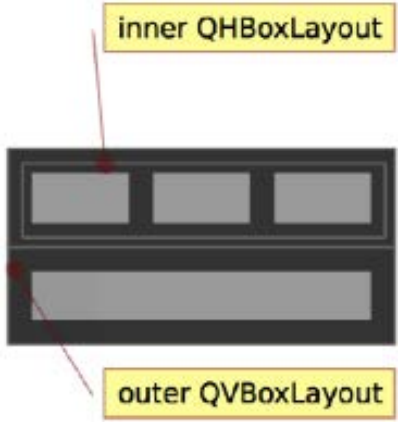
- **Stretch**
  - *Relative resize factor*
  - QBoxLayout::addWidget(widget, stretch)
  - QBoxLayout::addStretch(stretch)
  - QGridLayout::setRowStretch(row, stretch)
  - QGridLayout::setColumnStretch(col, stretch)
- **Contents Margins**
  - *Space reserved around the managed widgets.*
  - QLayout::setContentsMargins(l,t,r,b)
- **Spacing**
  - *Space reserved between widgets*
  - QBoxLayout::addSpacing(size)



Layout Terms

- **Strut**
  - *Limits perpendicular box dimension*
  - e.g. height for QHBoxLayout
  - *Only for box layouts*
- **Min, max and fixed sizes**
  - QWidget::setMinimumSize(QSize)
  - QWidget::setMaximumSize(QSize)
  - QWidget::setFixedSize(QSize)
  - *Individual width and height constraints also available*
- **Nested Layouts**
  - *Allows flexible layouts*
  - QLayout::addLayout(...)





Widgets

Size Policies

- **`QSizePolicy`** describes interest of widget in resizing

```
QSizePolicy policy = widget->sizePolicy();
policy.setHorizontalPolicy(QSizePolicy::Fixed);
widget->setSizePolicy(policy);
```

- One policy per direction (horizontal and vertical)
- Button-like widgets set size policy to the following:
  - may stretch horizontally
  - are fixed vertically
  - Similar to `QLineEdit`, `QProgressBar`, ...
- Widgets which provide scroll bars (e.g. QTextEdit)
  - Can use additional space
  - Work with less than `sizeHint()`
- `sizeHint()`: recommended size for widget

Widgets
Size Policies

Policy	sizeHint()	Widget
Fixed	authoritative	can not grow or shrink
Minimum	minimal, sufficient	can expand, no advantage of being larger
Maximum	is maximum	can shrink
Preferred	is best	can shrink, no advantage of being larger
Minimum Expanding	is minimum	can use extra space
Expanding	sensible size	can grow and shrink

Hands-on

- Lab 4: Layout buttons
  - Objective
  - Template code

Summary

- How do you change the minimum size of a widget?
- Name the available layout managers.
- How do you specify stretch?
- When are you allowed to call resize and move on a widget?

Widgets

- Common Widgets
- Layout Management
- Custom Widgets

Custom Widget
Creation

- It's as easy as deriving from QWidget

```
class CustomWidget : public QWidget
{
public:
explicit CustomWidget(QWidget* parent=0);
}
```

- If you need custom Signal Slots
  - add **Q_OBJECT**
- Use layouts to arrange widgets inside, or paint the widget yourself.

Custom Widget

Base class and Event Handlers

- Do not reinvent the wheel
- Decide on a base class
  - Often QWidget or QFrame
- Overload needed event handlers
  - Often:
    - QWidget::mousePressEvent(),
    - QWidget::mouseReleaseEvent()
  - If widget accepts keyboard input
    - QWidget::keyPressEvent()
  - If widget changes appearance on focus
    - QWidget::focusInEvent(),
    - QWidget::focusOutEvent()

Custom Widget

Drawing a Widget

- Decide on composite or draw approach?
  - If composite: Use layouts to arrange other widgets
  - If draw: implement paint event
- Reimplement QWidget::paintEvent() for drawing
  - To draw widget's visual appearance
  - Drawing often depends on internal states
- Decide which signals to emit
  - Usually from within event handlers
  - Especially mousePressEvent() or mouseDoubleClickEvent()
- Decide carefully on types of signal parameters
  - General types increase reusability
  - Candidates are bool, int and const QString&

Custom Widget

Internal States and Subclassing

- Decide on publishing internal states
  - Which internal states should be made publically accessible?
  - Implement accessor methods
- Decide which setter methods should be slots
  - Candidates are methods with integral or common parameters
- Decide on allowing sub classing
  - If yes
    - Decide which methods to make protected instead of private
    - Which methods to make virtual

Custom Widget

Widget Constructor

- Decide on parameters at construction time
  - Enrich the constructor as necessary
  - Or implement more than one constructor
  - If a parameter is needed for widget to work correctly
  - User should be forced to pass it in the constructor
- Keep the Qt convention with:

```
explicit Constructor(..., QWidget
*parent = 0)
```

Hands-on

- Lab 5: File chooser
  - Objective
  - Template code
- Lab 6: Compass widget
  - Objective
  - Template code

Object communication

Object
communication

- Signals & Slots
- Event Handling

Objectives

- How objects communication
- Details of signals & slots
- Which variations for signal/slot connections exist
- How to create custom signals & slots
- What the role of the Qt event loop is
- How Qt handles events

Object
communication

- **Between objects**

  Signals & Slots
- **Between Qt and the application**

  Events
- **Between Objects on threads**

  Signal & Slots + Events
- **Between Applications**

  DBus, QSharedMemory

Callbacks

General Problem

How do you get from "the user clicks a button" to your logic?

- Possible solutions
  - **Callbacks**
    - Based on function pointers
    - Not type-safe
  - **Observer Pattern (Listener)**
    - Based on interface classes
    - Needs listener registration
    - Many interface classes
- Qt uses
  - Signals and slots for high-level (semantic) callbacks
  - Virtual methods for low-level (syntactic) events.

Object
communication

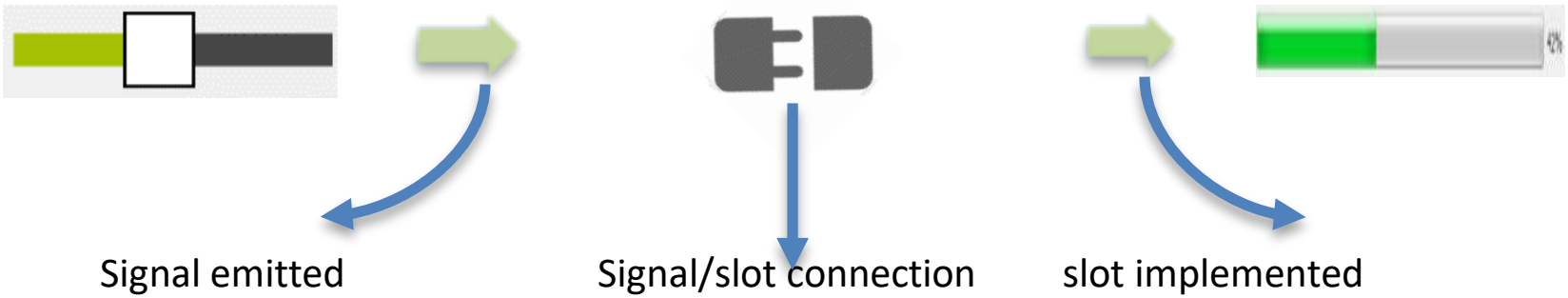
- Signals & Slots
- Event Handling

Connecting

signals and slots

```
void
Qslider::mousePressEvent(…)
{
    emit valueChanged(value)
}
```

```
void QProgressBar::setValue(int
value)
{
   mvalue = value
}
```



Connect Demo

```
QObject::connect(slider,SIGNAL(
valueChanged),progressbar,
SLOT(setValue))
```

EMERTXE

Connection
variants

- Qt 4 style:

```
connect(slider,SIGNAL(valueChanged(int)),
progressBar, SLOT(setValue(int)));
```

- Using function pointers (Qt5):

```
connect( slider, &QSlider::valueChanged,
progressBar, &QSpinBox::setValue );
```

  Demo

- Using non-member function:

```
static void printValue(int value) {...}
connect( slider, &QSignal::valueChanged,
&printValue );
```

  Demo

Custom slots

- File: **myclass.h**

```
class MyClass : public QObject
{
Q_OBJECT // marker for moc
// ...
public slots:
void setValue(int value); // a custom slot
};
```

- File: **myclass.cpp**

```
void MyClass::setValue(int value) {
// slot implementation
}
```

Demo

Custom signals

- File: **myclass.h**

```
class MyClass : public QObject
{
Q_OBJECT // marker for moc
// ...
signals:
void valueChanged(int value); // a custom signal
};
```

- File: **myclass.cpp**

```
// No implementation for a signal
```

- Sending a signal

```
emit valueChanged(value);
```

- Demo

Q_OBJECT - flag

- **Q_OBJECT**
  - Enhances QObject with meta-object information
  - Required for signals
  - Required for slots when using the Qt4 way
- **moc** creates meta-object information
  - moc -o moc_myclass.cpp myclass.h
  - c++ -c myclass.cpp; c++ -c moc_myclass.cpp
  - c++ -o myapp moc_myclass.o myclass.o
- **qmake** takes care of mocing files for you

Variations of Signal/Slot

Signal(s)	Connect to	Slot(s)
One	✓	Many
Many	✓	One
One	✓	Another signal

- Signal to Signal connection

```
connect(bt, SIGNAL(clicked()), this,
SIGNAL(okSignal()));
```

- **Not** allowed to name parameters

```
connect( m_slider, SIGNAL( valueChanged(
int value ) )
this, SLOT( setValue( int newValue ) ) )
```

Rules for
Signal/Slot

Can ignore arguments, but not create values from nothing. Eg:

Signal		Slot
rangeChanged(int,int)	✓ ✓ ✓	setRange(int,int) setValue(int) Update()
valueChanged(int)	✓ ✓ X	setValue(int) Update() setRange(int,int)
textChanged(QSting)	X	setValue(int)

Hands-on

- Lab 1: Select color
  - Objective
  - Template code
- Lab 2: Slider
  - Objective
  - Template code

Object
communication

- Signals & Slots
- Event Handling

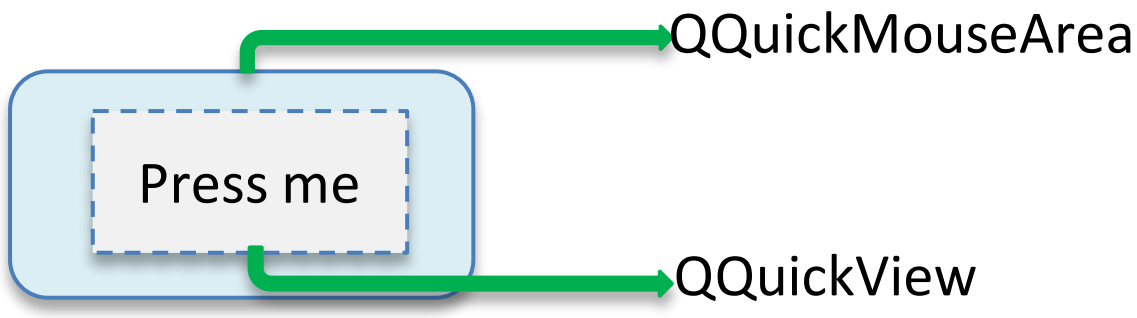
Event Processing

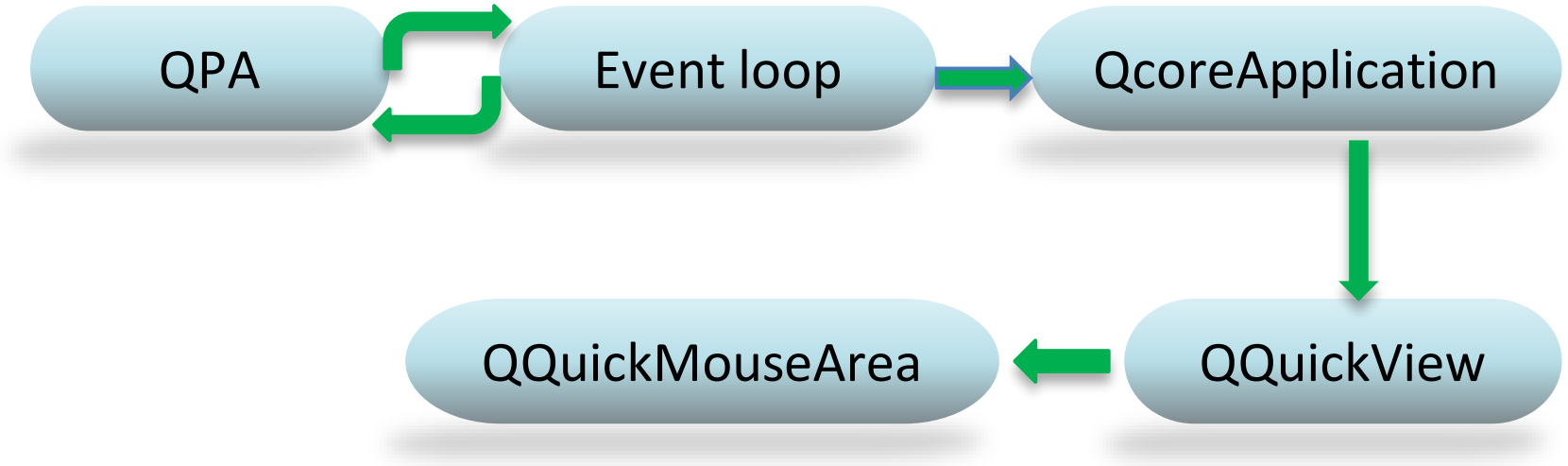
Qt is an event-driven UI toolkit

Qapplication::exec() runs the *event loop*

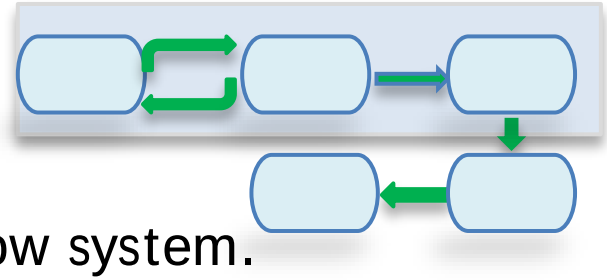
1. **Generate Events**
   - by input devices: keyboard, mouse, etc.
   - by Qt itself (e.g. timers)
2. **Queue Events**
   - by event loop
3. **Dispatch Events**
   - by QApplication to receiver: QObject
     - *Key events sent to widget with focus*
     - *Mouse events sent to widget under cursor*
4. **Handle Events**
   - by QObject event handler methods

Event Processing

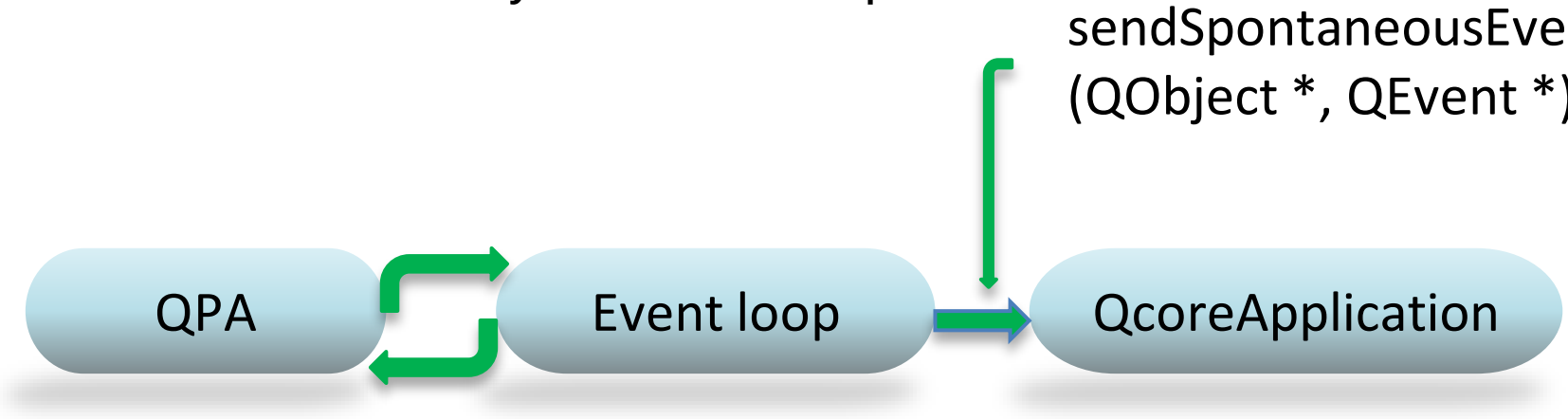




Event Processing

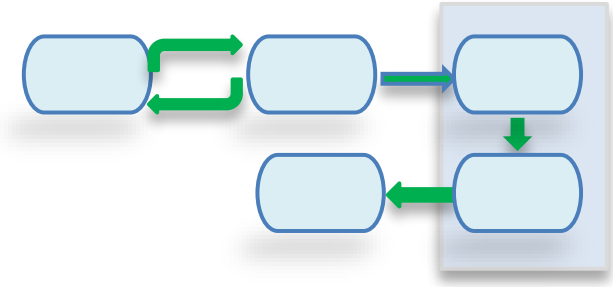


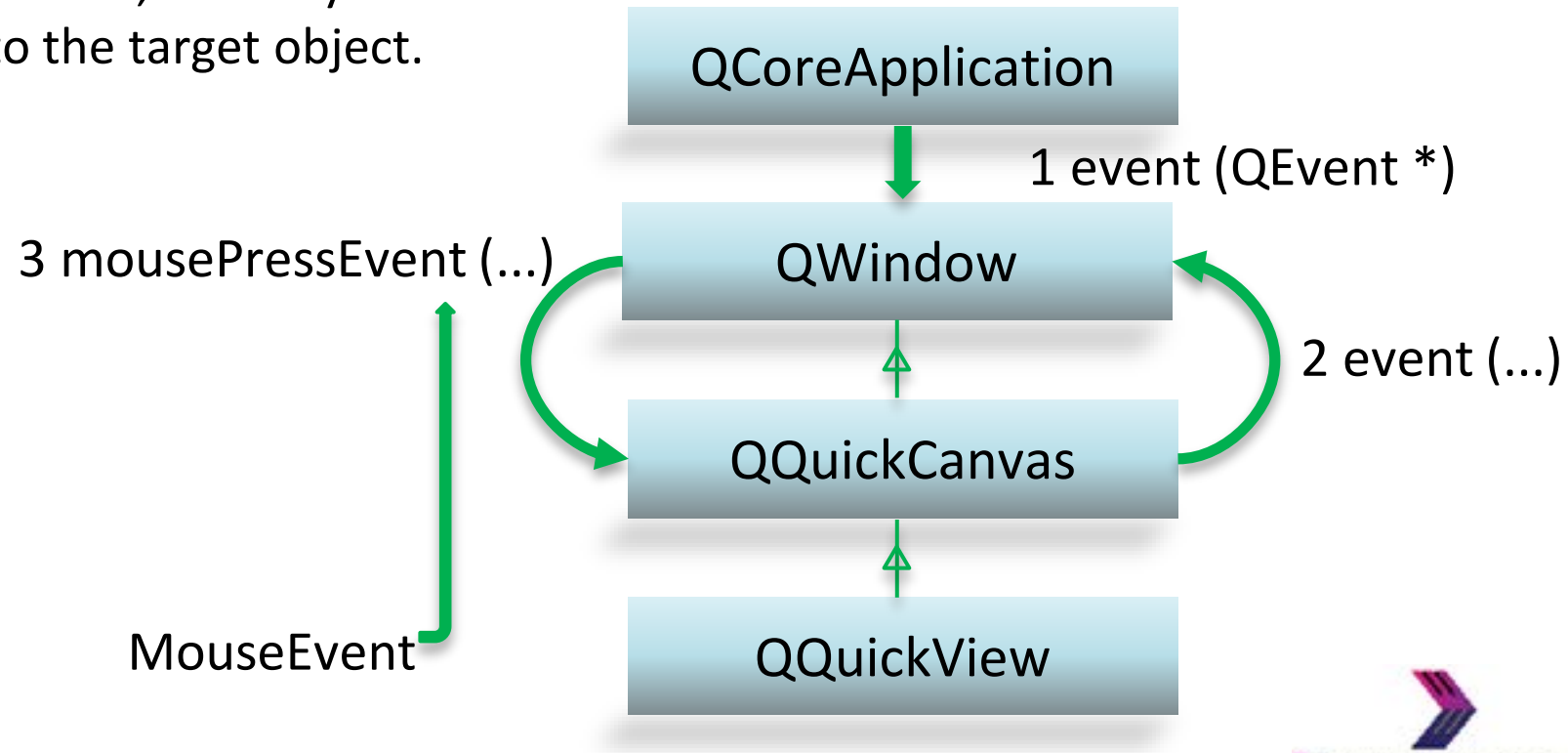
Spontaneous events are generated by the window system. They are put in a system queue and processed one after the other by the event loop



Event Processing

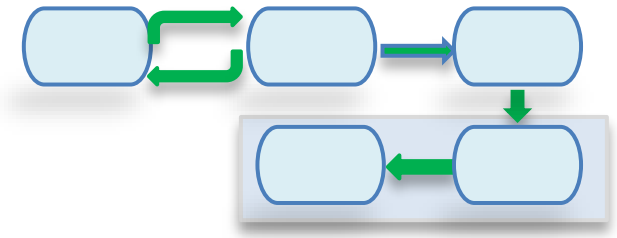
Posted events are generated by Qt or by the application. They are queued by Qt and processed by the event loop. ***Sent events*** are generated by Qt or by the application, but they are sent directly to the target object.



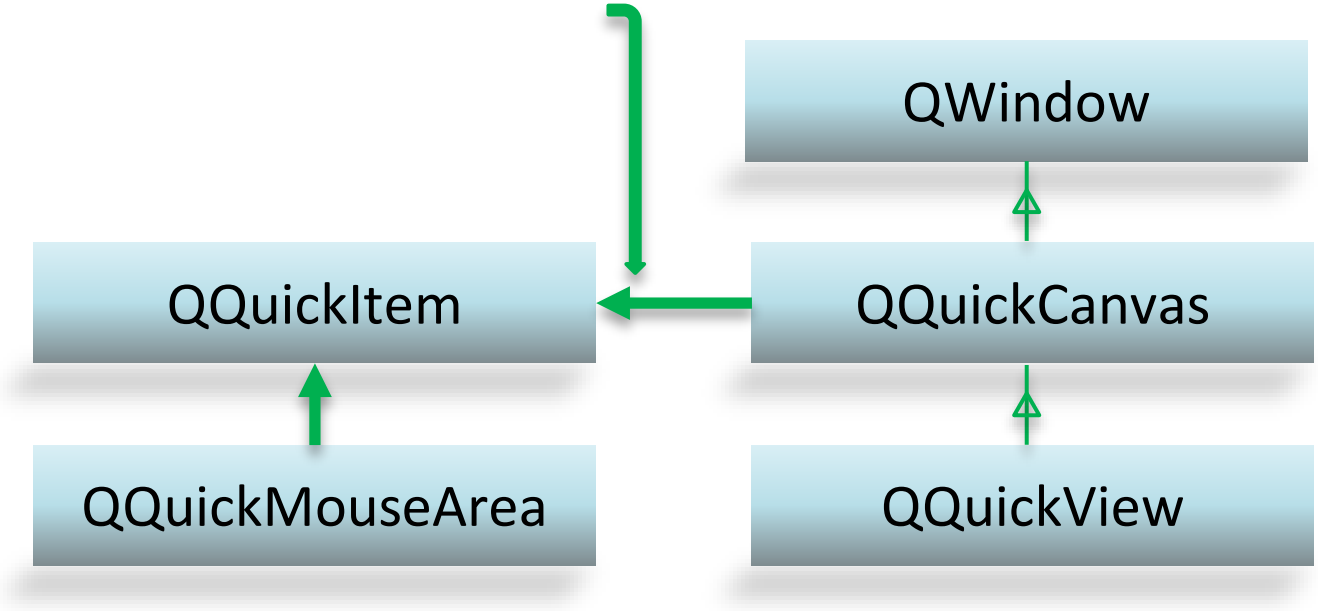


EMERTXE

Event Processing



mousePressEvent (QMouseEvent *)



EMERTXE

Event Handling

- QObject::event(QEvent *event)
  - Handles all events for this object
- Specialized event handlers for QWidget and QQuickItem:
  - mousePressEvent() for mouse clicks
  - touchEvent() for key presses
- Accepting an Event
  - event->accept() / event->ignore()
    - Accepts or ignores the event
    - Accepted is the default.
- Event propagation
  - Happens if event is ignored
  - Might be propagated to parent widget

Demo

Event Handling

- QCloseEvent delivered to top level widgets (windows)
- Accepting event allows window to close
- Ignoring event keeps window open

```
void MyWidget::closeEvent(QCloseEvent *event) {
  if (maybeSave()) {
    writeSettings();
    event->accept(); // close window
  } else {
    event->ignore(); // keep window
  }
}
```

Demo

Event & Signals

Signals and Slots are used instead of events

- To communicate between components.
- In cases where there is a well-defined sender and a receiver.
  - Eg: A button and a slot to handle clicks
- For some events, there is no sender in Qt
  - Eg: Redraw, keyboard and mouse events.
- To describe high level logic and control flow

Developers can create custom events if they need to

Summary

- How do you connect a signal to a slot?
- How would you implement a slot?
- How would you emit a signal?
- Can you return a value from a slot?
- When do you need to run qmake?
- Where do you place the Q_OBJECT macro and when do you need it?
- What is the purpose of the event loop
- How does an event make it from the device to an object in Qt?

THANK YOU